Assignment 3

By: Armando Herrera III

Abstract:

Here we use an SVM implemented in Assignment 2 and add FDA to reduce the number of samples required to train the SVM, decrease training time, and measure accuracy as we decrease the number of samples we train with, using FDA. We see with that reduction of the number of samples we train with, using FDA, it does indeed reduce training speed and, initial, only reduces testing accuracy slightly. When reducing the number of samples substantially, the accuracy is reduced substantially as well.

Introduction

Support Vector Machines

A Support Vector Machine (SVM) is a supervised learning model that analyzes data for classification and regression. In the field of machine learning, we know that SVMs are a robust method of classification and regression obtaining accuracy rival only to that of deep neural networks. An SVM, in the context of a classifier, is a binary classifier, so it classifies between one category or another. In a more mathematical description, an SVM create a hyperplane, a plane in n-dimensional space. This hyperplane separates the object's classes. An SVM can either have a hard or soft-margin. With a hard margin, when training, an object with a certain class isn't allowed be within the hyperplane's margins while, in a soft margin, the margins are more flexible. Sometimes, when a dataset isn't linearly separable the SVM will perform poorly. To fix this a kernel is used, it increases the dimensionality of the hyperplane, allowing for the separation of the classes; this is called the kernel trick. There are various methods of finding this hyperplane that separates all classes. One way described in the SVM Wikipedia page uses the hinge loss and reduces the SVM to a quadratic programming problem. More modern methods exist such as, Sub-gradient descent where Stochastic Gradient Descent (SGD) or Gradient Descent (GD) are adapted with the quadratic programming problem, in this paper, I do something like this.

In terms of the Kernel Trick, there are various methods to perform this function. In an SVM, the inner product is computed between a weights vector and an input vector with a dimensionality n then a bias is added. The kernel function increases the dimensions in which the weights and the input vector operate by replacing the dot product with a function of the form:

$$k(x, x') =< \varphi(x), \varphi(x') >$$

The kernel most in use, for a good reason, is called the Radial Basis Function (RBF) kernel and is defined as:

$$k(x, x') = \exp(-\gamma \left|\left|x - x'\right|\right|^2)$$

Where γ is a tunable floating-point hyper-parameter between 0 and 1. For when there are many samples and in cases where there is large dimensionality there exists approximations for kernel functions where a function $z(x)$ is created where:

$$k(x, x') = < \varphi(x), \varphi(x') > \approx < z(x), z(x') >$$

For the RBF kernel there are two main functions that can be used in place of the dot product, one is called the Nyström method and the other is called the kitchen sink method and uses a Monte Carlo method along with Fourier transformation. Since I use the kitchen sink method in this project, I will go into it in more detail in the model section.

To account for multiple classes there, exist three main methods of applying and having an SVM classify. One way, and the one used in this project, is called "one vs. all" way where an SVM is created for each class and each class is pitted against all other classes. In other words, each SVM classifies whether it is that class or it is not that class. The max argument is then taken to be the most likely class. The second way "one vs. one" way where an SVM is created for each possible combination of class and the argmax of an aggregated score is used to classify. The third method is called a Directed Acyclic Graph (DAGSVM).

Why matrix multiplication makes for a perfect SVM

One of the reasons that a "one vs. all" scheme is used to create the SVMs is because of our ability to generalize and have all SVM's dot, or kernel, operations done in one fell swoop. Here is a proof:

proof. A single SVM's hyperplane is mathematically defined as:

$$\vec{w} \cdot \vec{x} - b = 0$$

Where $\vec{w}$ is the weights vector, $\vec{x}$ is the input vector, and b is a bias scalar. Let's say that there are 3 classes we want to differentiate from, and we go with a "one vs. all" approach we would then create three different SVMs.

$$\overrightarrow{w_1} \cdot \overrightarrow{x_1} - b_1 = 0$$

$$\overrightarrow{w_2} \cdot \overrightarrow{x_2} - b_2 = 0$$

$$\overrightarrow{w_2} \cdot \overrightarrow{x_2} - b_2 = 0$$

We can reduce the dot product operations to its base components.

$$\sum_{i=1}^{n} w_{1i} x_{1i} - b_1 = 0$$

$$\sum_{i=1}^{n} w_{2i} x_{2i} - b_2 = 0$$

$$\sum_{i=1}^{n} w_{3i} x_{3i} - b_3 = 0$$

Matrix multiplication is defined as:

$$c_{ij} = \sum_{k=1}^{n} a_{ik} b_{kj}$$

Notice since there is only one value for the index in the summations for the SVM that are not part of the summation, we can switch around the indexes to mimic the Matrix Multiplication by changing the order of operations.

$$\sum_{i=1}^{n} w_{1i}x_{i1} - b_1 = 0$$

$$\sum_{i=1}^{n} w_{2i}x_{i2} - b_2 = 0$$

$$\sum_{i=1}^{n} w_{3i}x_{i3} - b_3 = 0$$

Notice again, since the enumeration of the SVM are increasing from 1 to 3 we can then represent the series of SVM in terms of Matrix Multiplication.

$$\sum_{i=1}^{n} w_{ji}x_{ij} - b_j = 0 | \, j \in [1,3] \cap Z^+$$

□

Fisher Linear Discriminant Analysis (FDA)

FDA is a method used to find a linear combination of features that separate two classes. This is used as a linear classifier and for dimensionality reduction before further classification. FDA starts with an object function.

$$J(w) = \frac{(\tilde{\mu}_1 - \tilde{\mu}_2)^2}{\tilde{S}_1^2 + \tilde{S}_2^2}$$

Where S_1 and S_2 are defined as the scatter of each class. The objective function is then used to solve for a, v, line that best separates the classes by solving a generalize eigenvalue problem.

$$S_B v = \lambda S_W v$$

The solution for which is:

$$v = S_W^{-1}(\mu_1 - \mu_1)$$

This method has the same problem as the SVM, the FDA is a binary classifier not a multi-class classifier. There is a method to generalize FDA into Multiple Discriminant Analysis (MDA) but in this project I stuck with using many FDAs with the “one vs. all” scheme.

Stochastic Gradient Descent (SGD)

SGD is an iterative optimization algorithm. SGD is derived from Gradient Descent (GD). Unlike GD, SGD, instead of calculating the gradient explicitly, calculates an estimate of the gradient.

$$w := w - \eta \nabla Q_i(w)$$

Where w are the weights we are optimizing, η is the learning rate, and $\nabla Q_i(w)$ is the approximation of the gradient. The approximation for the gradient is, typically, calculated, not with numerical differentiation, but with back-propagation. Back-propagation is a, quick, method of calculating both the value and a gradient in one fell swoop using the chain rule from Calculus; Automatic Differentiation is generalization of back-propagation.

PyTorch

PyTorch is an open source machine learning library for Python based on Torch. It is used in many applications such as Computer vision and Natural Language Processing. It was developed and is maintained by Facebook and an open-source community around it. It is well known for being very dynamic in that you can perform any differentiable operations and you can use PyTorch to easily backpropagate and calculate gradients using something known as a Gradient Tape. A Gradient Tape is a sort of tape that records all operations performed on a designated tensor, matrix, vector, or scalar. This makes it one of a kind and this Gradient Tape is yet to be implemented anywhere else, including MATLAB. TensorFlow only just implemented a Gradient Tape in their 2.0 release in September. This Gradient Tape tool is a strong tool for researchers!

Model

FDA

Here, an FDA based model is used make a rough calculation of linear separation between classes in a "one vs. all" scheme. With this scheme in mind, we can make some optimization to the calculations via memorization. The scatter matrix for each class is first calculated.

$$S_i = \sum_{x_j \in Class\ i} (x_{ij} - \mu_i)(x_{ij} - \mu_i)$$

Where x_{ij} is the sample from the ith class and the jth sample in that class and μ_i is the mean of that class. Then S_w, the within classes scatter is calculated by summing all the scatter matrices.

$$S_w = \sum_{i=1}^{c} S_i$$

Where c is the number of classes. Then, using the inverse of S_w the discriminant line is calculated on a per class basis.

$$v_i = S_w^{-1}(\mu_i - \frac{1}{c-1}(\sum_{j=0}^{i-1} \mu_j + \sum_{j=i+1}^{c} \mu_j))$$

Then to calculate the distance from the line on a per class basis.

$$y_i = v^T c_i^T$$

The closest percentage of points used in the SVM.

SVM

The SVM model used is the one described in the proof in the SVM section of the Introduction where a weight matrix, $w \in R^{m \times n}$, and a bias vector, $b \in R^m$, is defined. Let m be the number of classes and n be the number of dimensions the kernel outputs. We then, according to our kernel approximation, define a random weights matrix offset, $w_{offset} \in R^{m \times n} \cap \left[-\sqrt{2\gamma}, \sqrt{2\gamma}\right]$, and a random bias vector offset, $b_{offset} \in R^m \cap [0, 2\pi]$, of the same size. The weight was randomly initiated with real numbers in the interval $[-\sqrt{2\gamma}, \sqrt{2\gamma}]$ and the bias in the interval $[0, 2\pi]$. The kernel function is as follows.

$$z(x) = \frac{\sqrt{2}}{N} \cos(x w_r + b_r)$$

Where N is the number of output kernel dimensions and x is the input sample. The SVM output operation is then:

$$f(x) = wz(x) + b$$

To train the SVM SGD was the optimization algorithm of choice. Since the output of the kernel function is not differentiable natively the cross-entropy loss function was used. It is defined as:

$$H(p, q) = -\sum_i p_i log q_i$$

Where H is the cross entropy, q is a binary indicator and p is the predicted probability.

More details on the model and implementation can be obtained via the GitHub project.

Methods

While testing we noticed that tuning hyper-parameters was an incredibly long and tedious task, so Optuna, an advanced hyper-parameter optimization library, was used to tune hyper-parameters not otherwise mentioned. On the first run all hyper-parameters were tuned as to find the best possible accuracy the SVM could have gotten with all data. Subsequently, only the learning rate, the gamma, and the number of kernel dimensions were tuned using the already optimal number of epochs. A set of benchmarks were then done with varying cutoff percentages were done. These benchmarks started with 25 trials of hyper-parameter optimization using Optuna, followed by a timed training with the found optimal parameters. Testing accuracy and training time was measured and recorded. Since the cutoff rates varied the number of samples used for training consequently varied, this was recorded as well.

The benchmark starts with a cutoff rate of 0, so all samples are used to train the SVM. Followed by an optimization, training, testing, and recording at every 0.2 interval until reaching 0.8. The data tested was the MNIST handwritten digits dataset from the National Institute of Standards and Technology.

Results

accuracy	cutoff	Number of samples	time (s)
95.49	0	60000	203.6372
91.51	0.2	35044	116.7968
69.81	0.4	15331	50.37569
56.16	0.6	4609	14.94336
33	0.8	538	2.090637

As the number of samples decreased so did the accuracy and the training time. Between the first two entries there was a substantial decrease in training time with only a slight decrease in accuracy. After which the accuracy decreased drastically.

Conclusion

With the use of PyTorch and Optuna we can successfully create an SVM with a high accuracy with the MNIST dataset. With the use of FDA, we can successfully half training time with only a slight decrease in accuracy and cut it by a third with a reduction of accuracy to about 70%. The use of a kernel function, while it increased our accuracy substantially, it has the potential to decrease generalization which we did not measure as generalization is a difficult metric to measure. A foreseeable avenue is the pursuit of just that, a measure of a metric of the generalization with and without the kernel function and with and without the FDA pruning.